第8章 アナログ・データ・キャプチャの製作

── USB Mass Storage Classに対応した簡易計測器

河野 崇

ここでは，USBを介してパソコンと接続できる簡易計測器の設計事例を紹介する．ADuC7026のA-Dコンバータからデータを取り込む．USB Mass Storage Classに対応しており，パソコン側のドライバ・ソフトウェアを開発しなくても使用できる．（編集部）

最近の組み込み機器に要求される機能の一つとして，パソコンとの接続機能があります．

ひと昔前はシリアル（EIA-232）接続が主流でした．この規格は，もともとモデムとパソコンを接続するために策定されたものであり，リモート・ホストとのキャラクタ・ベースのデータ通信に最適化されています．汎用性が高く直感的に理解しやすいため，簡単に制御ソフトウェアを書くことができます．多くの場合，組み込み機器をパソコンから制御するという用途ではこれで十分です．しかし，転送速度が低いため，高い応答性が必要だったり，大量のデータを高速にやりとりするという用途には不向きです．

これに対して，最近のパソコンでよく使われているUSBは，制御にも高速データ転送にも適した規格です．しかし，一部の規格化された機能以外については，OSに依存した専用のドライバ・ソフトウェアを用意する必要があり，開発に負担がかかるという問題があります．

本稿では，USB固有の情報を知らなくても，オープンな規格に準拠した方法でUSBを活用する手法を提案します．USB Mass Storage Classへ対応することで，計測器やアクチュエータなどの制御機能とデータ転送機能を持つ機器をUSB経由で接続します．

Mass Storage ClassはUSB経由でハード・ディスクやフロッピー・ディスクといった外部記憶装置を接続するための規格で，多くのOSで標準的にサポートされており，特別なドライバがなくても利用できます．OSは，コマンド/ステータス用およびデータ転送用のパーティションをもつ外部記憶装置としてデバイスを認識します．

制御用ソフトウェアは，POSIX規格で定義されている入出力システム・コール`open()`，`read()`，`write()`，`close()`を用います．デバイス・ファイルにアクセスすることにより，デバイスの制御やデータの入出力を行います．デバイスは仮想的に，あるいは物理的に制御用やデータ用の記憶領域（パーティション）を用意し，それらに対してパソコンがUSB経由で送ってくるMass Storage Classのコマンドを解釈・実行します．制御用パーティションに対する書き込みはコマンド発行，読み出しはステータス要求として解釈します．もちろん，データ用パーティションに対する読み書きはデータの読み書きと解釈します．

実装例として，ADuC7026内蔵のA-Dコンバータ（2チャネル）からアナログ・データを取り込んでパソコンに転送する簡易計測器を作成しました．

OSは，ストレージ機器に対してキャッシュ機構を持っています．`open()`システム・コールでオープンする限り，オプション・フラグ（`O_DIRECT`と`O_SYNC`）を設定することでキャッシュを回避することができます．しかしここでは，アクセス方法の選択肢を広げ，あらゆる環境下でOSのキャッシュ機構を避けることを意図して，メディア・ロック機構のないUSBフロッピー・ディスクをエミュレートしています．こうすることで，ディレイは避けられないものの，ライト・キャッシュを自動的に回避できます．

1. 簡易計測器のシステム構成

設計した簡易計測器のブロック・ダイヤグラムを**図1**に

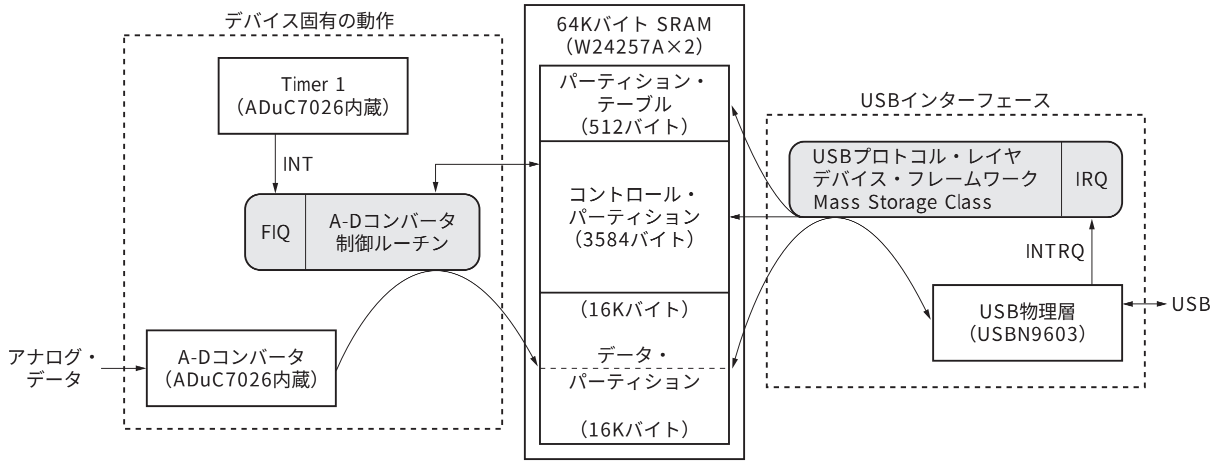


図1　設計した簡易計測器のブロック・ダイヤグラム

USBインターフェースとデバイス固有の動作(今回はA-Dコンバータからのデータ取り込み)の独立性を高め，機能追加(例えばD-Aコンバータへのデータ出力など)をUSBインターフェースに手を加えることなく実現できるようにするために，64Kバイトの外付けSRAMを用いて物理的にメモリ・パーティションを用意している．

示します．

USBインターフェースとデバイス固有の動作(今回はA-Dコンバータからのデータ取り込み)の独立性を高め，機能追加(例えばD-Aコンバータへのデータ出力など)をUSBインターフェースに手を加えることなく実現できるようにするために，64KバイトのSRAMを用いて物理的にメモリ・パーティションを用意しています．

● USBインターフェース部

図1の右側にUSBインターフェースの構成が示されています．

USB物理層については，米国National Semiconductor社の「USBN9603」に任せ，プロトコル層(USBトランザクションの処理)，デバイス・フレームワーク(USBデバイス・リクエストの処理)，Mass Storage Classの処理をソフトウェアで行います．

ソフトウェアでは，USBN9603の割り込み要求信号(INTRQ)によって，ADuC7026のIRQ割り込みとして呼び出されます．

● アナログ・データ取り込み部

図1の左側にデバイス固有の動作(A-Dコンバータからのデータ取り込み)が示されています．

このソフトウェアは，ADuC7026内蔵のタイマ(Timer 1)の割り込みによって呼び出されます．Timer 1にはサンプリング周波数がセットされており，呼び出されるたびに制御パーティションをチェックし，コマンドの解釈を行います．

データの取り込みが指示されれば，呼び出されるたびにA-Dコンバータからデータを読み出して，SRAMのデータ・パーティションに格納します．また，必要に応じてステータスを制御パーティションに書き込みます．この処理はサンプリング周波数ごとに確実に行わなければならないため，USBN9603の割り込み信号よりも優先度の高いFIQ割り込みとして呼び出しています．

具体的には，制御パーティションは0バイト目がCH0へのコマンド，1バイト目がCH1へのコマンド，2バイト目がコマンド・キューとなっています．

今回は説明を簡単にするため，データ取り込み開始コマンド(‘G’)と中止コマンド(‘N’)だけを用意しました．コマンド・キューは，CH0，CH1へのコマンド・バイトに有効なコマンドが書き込まれたことを示すためのものです．デバイス側は，このバイトに‘*’が書き込まれたことを認識すると，これを‘ ’(スペース文字)でクリアすると同時に，CH0，CH1のコマンド・バイトを解釈・実行します．

データ・パーティションとしては，CH0，CH1用にそれぞれ16Kバイトずつのパーティションを用意しました．‘G’コマンドが与えられると，対応するチャネルのパーティションの先頭から順次，A-Dコンバータからのデータを格納していきます．末尾に到達したら対応するコマンド・バイトに動作完了のマークとして‘#’を書き込み，終了します．

パソコン側はコマンド発行後，定期的に制御パーティシ

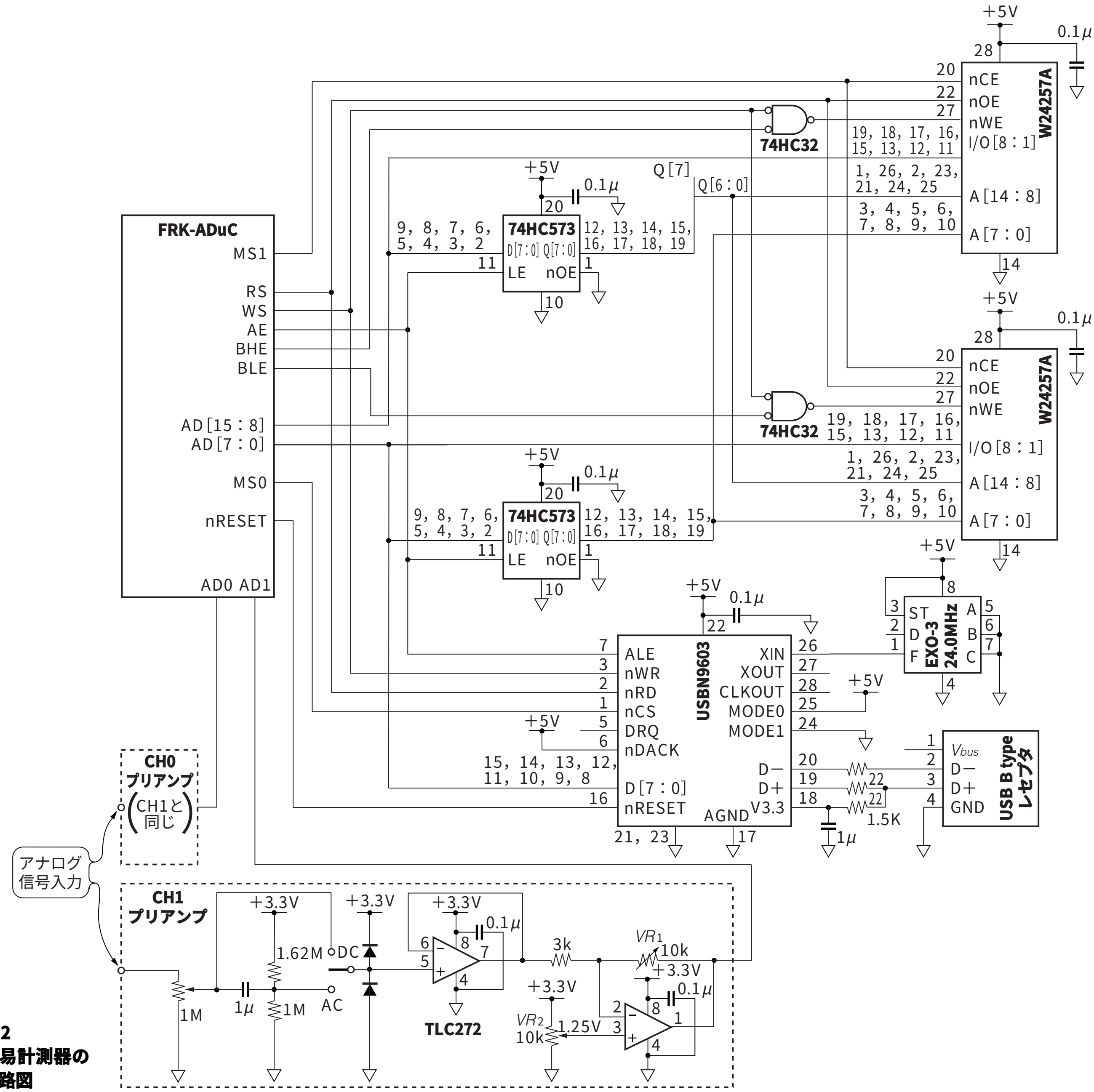


図2
簡易計測器の回路図

ョンをチェックし，‘＃’が書き込まれるのを待つことでデータ取り込みの完了を知ることができます．データそのものは，それぞれのチャネルに対応するパーティションを読み出すことで得ることができます．

2．ハードウェアの設計

設計した簡易計測器の回路図を**図2**に示します．

ADuC7026は，GPIOのP0.1，P0.2，P2.1～P2.7，P3，P4をコンフィグレーション2に設定（GPxCONレジスタ，x＝0/2/3/4）することで，アドレス・データ多重の16/8ビットのメモリ・バスを使用することができます．このバス経由でアクセスできるメモリ空間は，128Kバイトずつの4領域に分けられており，各領域ごとにイネーブル信号（MS[3：0]）が用意されています．USBN9603および外付けSRAMをこのバスに接続し，前者を領域0に，後者を領域

1に割り当てました．

● USBインターフェース部

USBN9603は，8ビットのアドレス多重バスをサポート（MODE1＝'0'，MODE0＝'1'に設定）しています[2]．こうすることでADuC7026のメモリ・バスに直結することができます．

ADuC7026側では，XM0CONレジスタのビット1を'0'とすることで領域0のアクセス・モードを8ビットに設定できます．

USBN9603の割り込み要求信号（INTRQ）をADuC7026の外部割り込み入力ピンIRQ0に接続し，USBN9603からの割り込みを受け付けられるようにしておきます．

● 外付けSRAM部

外付けSRAMには，台湾Winbond Electronics社の「W24257AK-15」を使用しました．これは一般的な8ビット幅の256KビットSRAM[3]で，ルネサス テクノロジーの「HM65256」やNECの「μPD43256」とピン配置互換です．

アドレス・ラッチとして，74HC573を2個使用しました．ALE出力によってアドレスを分離し，SRAMに供給しています．BHE，BLE出力は，16ビット・バス上で8ビット単位のアクセスを行うための信号で，それぞれ上位バイト，下位バイトへのアクセスが必要となるときにアクティブになります．16ビットと8ビットのアクセスを区別する必要があるのは書き込みだけですから，ライト・ストローブ信号WSとの論理積をSRAMに供給し，リード・ストローブ信号RSはそのまま接続します．

● バス・タイミングの調整

ADuC7026には，メモリ・バスのアクセス・タイミングを設定するレジスタがメモリ領域ごとに用意されています（XMxPAR，x＝0/1/2/3）．ADuC7026のメモリ・タイミング・チャートとUSBN9603，SRAMのタイミング・チャートを比較してタイミング設定を決定し，実機でチェックしながら調整したところ，USBN9603については0109h，SRAMについては0201hが最速の設定でした．

USBN9603のアクセス・タイミングは，アドレス関連のアクセス時間は早いものの，リードでは読み出し時間が，ライトでは書き込みのあと次のアドレス設定が可能になるまでの時間が250nsと長いことに注意が必要です[2]．とくに，ライトのあとの待ち時間はCPUクロックで約10サイクル必要ですが，XMxPARレジスタでは1クロックまでしか設定できないため，場合によってはプログラミング上の配慮も必要となります．

● アナログ入力部

アナログ入力については，CH0とCH1のそれぞれに簡単なアンプを用意しました（**図2**参照）．

A-Dコンバータの入力範囲は0V～2.5Vなので，このアンプによりレンジ調節を行います．入力信号はまずアッテネータを通って振幅調整されます．次に，AC接続の場合はカップリング・コンデンサとバイアス抵抗を経由し，DC接続の場合は，直接OPアンプに入力されます．OPアンプの1段目はボルテージ・フォロワを構成しており，インピーダンス変換を行います．2段目は反転増幅器で，1.25V（VR_2の分圧比で決定される電圧）を中心として，（VR_1の定数）/3k倍に増幅します．すべて単一電源で構成されており，負の電圧を計測したい場合はAC接続にしなければなりません．また，OPアンプは単一電源3.3Vで動作するものであれば何でもかまいません（ただし，ADuC7026のA-Dコンバータの精度を生かすには，それに見合った部品を使う必要がある）．反転増幅器を通っているので，得られる値はスケール値（4096）から引いて正しい値になります．

3．ソフトウェアの開発

ソフトウェアのブロック図を**図3**に示します．

● 割り込み処理

ARM7TDMIシリーズの割り込みにはIRQとFIQの2種類があります．後者は割り込みルーチンで使用できるレジスタ数などに制限を加えることで割り込み信号検出からルーチン・コールまでの処理を軽減し，レイテンシを抑えています．また，ARM7TDMIの割り込み禁止フラグについては，CPSRというCPUレジスタにIRQ用（Iビット）とFIQ用（Fビット）が別々に用意されています．これらの禁止フラグは割り込みルーチンが呼ばれるときに自動的にセットされますが，IRQ割り込み時はIビットのみ，FIQ割り込み時にはIビットとFビットの両方がセットされます．つまり，IRQ割り込み処理中でもFIQ割り込み処理を受け付けることができます．したがって，IRQとFIQを使い分

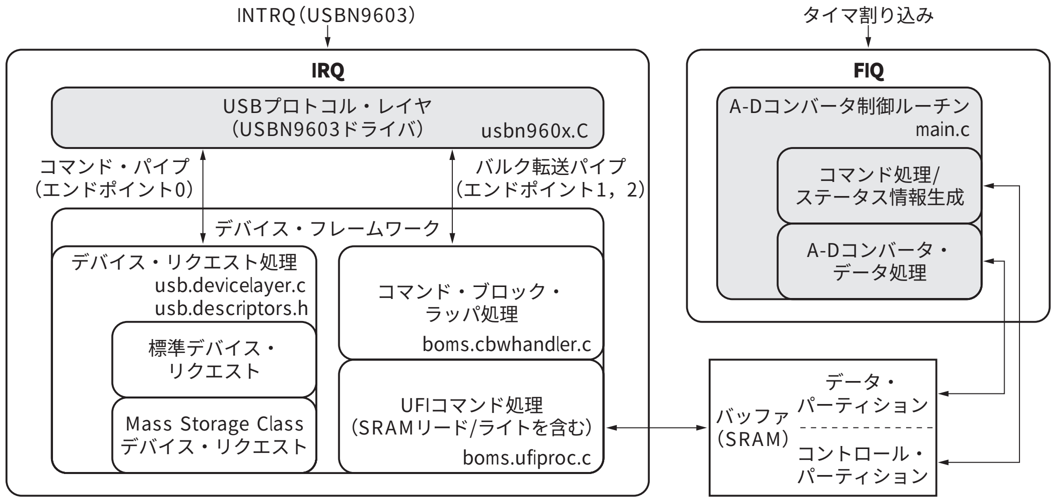


図3
ソフトウェアのブロック図

けることで，CPSRにアクセスしなくても，割り込み処理中により優先度の高い割り込みを受け付けることができます．

データのサンプリング処理はサンプリング周波数ごとに必要な処理なので，FIQ割り込み処理にします．USB周りは処理が複雑になる場合もあり，それほど時間の制約も厳しくないため，IRQ割り込み処理にします．

Timer 1にデータのサンプリング間隔を設定し，周期的に割り込みを発生させます．具体的にはT1LDレジスタにカウント値を，T1CONの下位4ビットに分周比を，ビット7とビット6に‘1’をセットします．この割り込み信号でFIQを発生させられるように設定します（FIQENレジスタに00000008hを書き込む）．

USBN9603のINTRQ出力は，外部割り込み入力ピンIRQ0に接続してあるので，これによってIRQ割り込みを発生させられるように設定します（IRQENレジスタに00008000hを書き込む）．IRQENやFIQENレジスタでは，新たにセットしたいビットのみを立てて書き込むことで許可ビットを追加できる点が便利です．逆に許可ビットをクリアしたい場合は，IRQCLR，FIQCLRレジスタにクリアしたいビットのみを立てて書き込むことになります．

● A-Dコンバータの制御

A-Dコンバータ制御ルーチンでは，まず，FIQSTAレジスタをチェックします．このレジスタはFIQENレジスタで割り込みを許可している信号のうちのどの信号がアクティブになっているかを知らせてくれます．対応するビットが‘1’になっていればアクティブになっているので，このレジスタと00000008hの論理積が‘1’ならば，Timer 1からの割り込みと判断します（今回はFIQ割り込みはTimer 1からのものだけなので，この処理はなくても動作する）．

次に，T1CLRIレジスタに書き込みを行い，割り込み入力をクリアします．これを行わないと，割り込み処理終了後，割り込み処理がまたすぐ始まってしまいます．

次にコマンド・パーティションのコマンド・キュー・バイトをチェックします．ここに‘*’が書き込まれていれば，有効なコマンドであるCH0，CH1のコマンド・バイトがそれぞれにセットされているので，コマンド内容をチェックします．‘G’ならば該当チャネルのデータ収集を開始し，‘N’ならば中止します．これら以外ならば何もしません．

具体的には，データ収集開始の場合は変数ADC_CHx_EN（x＝0/1）に‘1’をセットし，中止の場合は‘0’をセットします．同時に，データ・メモリへのポインタ（変数adcChxCurrent，x＝0/1）をメモリ領域の先頭（ADC_CHx_BUFHEAD，x＝0/1）にセットします．そして，ADC_CHx_ENが‘1’の場合は該当チャネルからのデータを読み出し，メモリへ格納し，ポインタをインクリメントします．ポインタがメモリ領域の最後（ADC_CHx_BUF-TAIL－1，x＝0/1）まで達したらデータ収集を完了し，該当コマンド・バイトに‘#’（完了ステータス）を書き込みます．

● USBインタフェース処理

USBインターフェース・ルーチンでは，A-Dコンバータ制御ルーチンの場合と同様，まずIRQSTAレジスタをチェ

ックし，外部割り込み0からの割り込みであることを確認してからUSBN9603の制御ルーチン（`USBN960x_int()`）を実行します．制御ルーチンはusbn960x.cに記述されており，**図3**では左上のUSBプロトコル・レイヤに相当します．USBN9603の制御方法についてはデータ・シート以外にも多数の参考資料(4), (5), (6)があるので，詳細はそちらを参考にしていただくとして，ここでは概略だけを解説します．

USBでは，一つのデバイスがエンドポイントというデータを送受信するための「出入り口」を，機能ごとに用意します．エンドポイントを選んでデータをやりとりすることで，パソコンは使用したい機能ブロックと直接，通信することができます．

USBN9603には，EP0～EP6の7個のエンドポイントが用意されています．EP0はUSB規格(7)（USBN9603はUSB Rev. 1.1対応）で必須とされているコントロール・エンドポイントで，8バイトのFIFOを備えています．これはUSB規格で指定されているとおり入出力兼用です．FIFOも入出力兼用で，EPC0レジスタによって方向を切り替えて使用します（入出力兼用なのはコントロール・エンドポイントのみ）．EP1，EP3，EP5は送信専用（USBのIN方向に対応）のエンドポイントで，64バイトのFIFOを備えています．EP2，EP4，EP6は受信専用（USBのOUT方向に対応）で，同じく64バイトのFIFOを備えています．

制御ルーチンのしごとは，なぜ割り込みがかかったのかを判定し，それに対応した指令をUSBN9603に与えることです．**図4**に，USBN9603のレジスタのうち，制御ルーチンでよく使うものを示します．すべて8ビットです．

MAEVレジスタには，割り込み発生の理由がおおまかに示されているので，まずこれをチェックします．

RX_EVビットが立っていれば，データの受信に関係する状態が変化したということです．どのチャネルで，何が起こったのか（受信が成功した，エラーが発生したなど）を知るためにRXEVレジスタ，さらにRXSx（x＝0/1/2/3）レジスタをチェックすることになります．

TX_EVビットが立っていればデータ送信の完了（あるいは未完了）を意味するので，TXEVレジスタやTXSx（x＝0/1/2/3）レジスタをチェックします．

NAKビットは，受信エンドポイント（EP2，EP4，EP6）においてFIFOがいっぱいでデータを受け取ることができなかった場合や，送信エンドポイント（EP1，EP3，EP5）

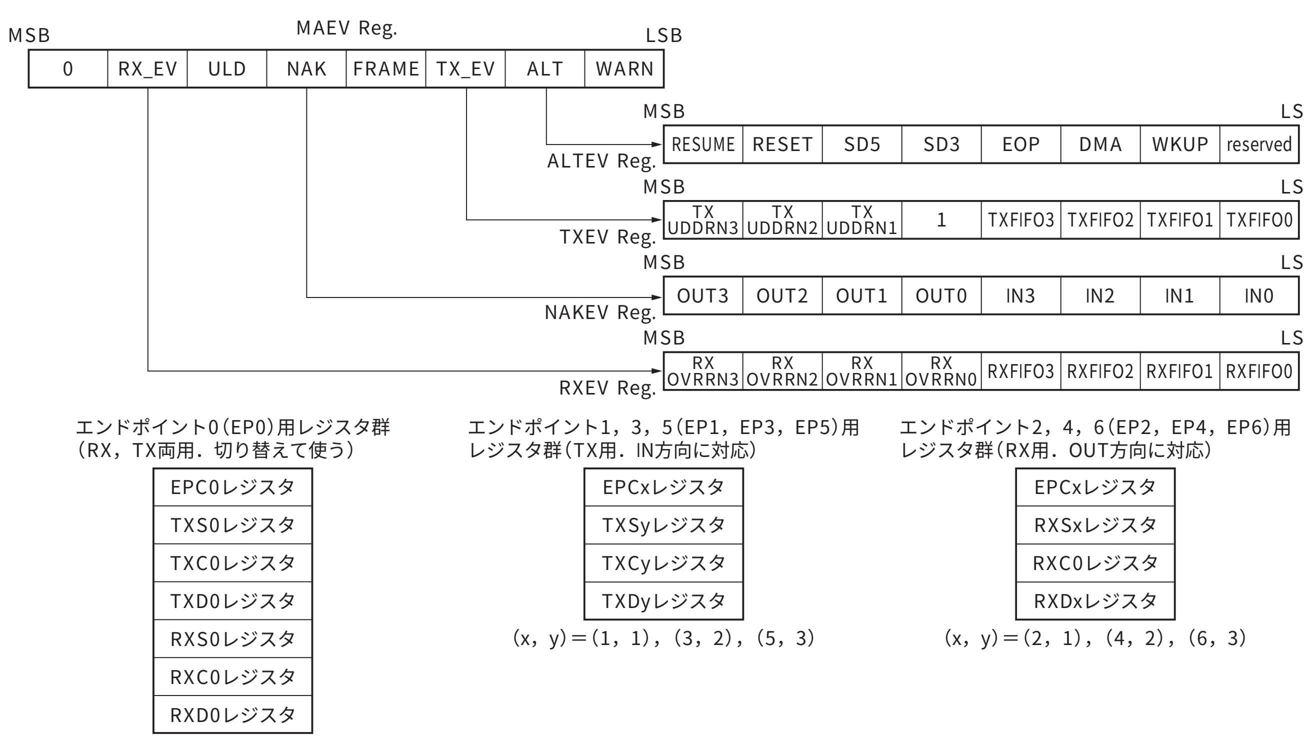


図4　USBN9603のレジスタ（一部）
制御ルーチンでよく使うものを示している．

においてFIFOが空だったためにパソコンからのデータ読み出し要求にこたえられなかった場合に‘1’になります．どのチャネルで発生したのかはNAKEVレジスタによって知ることができます．基本的にこのレジスタで重要なのはOUT0（ビット4）のみです．このビットはEP0の受信方向でNAKが発生したことを意味します．パソコンからのコマンドが来ているのに受け取れていないということなので，送信中でも中止して受信モードに切り替える必要があります．

ALTビットが立っていた場合は，パワー・マネージメント関連とバス・リセット関連のレジスタであるALTEVレジスタをチェックします．本機のようにセルフ・パワーのデバイスの場合は，パワー・マネージメントが必須というわけではありません．バス・リセット（ビット6）のみをチェックすれば十分です．バス・リセットが発生した場合は，電源投入時と同じ状態に戻らなければなりません．

USBN9603の制御ルーチンはこれらのレジスタを操作し，各エンドポイントのデータ送受信を行います．これについてはファイルusbn960x.cに記述されています．

● 標準デバイス・リクエストへの対応

USBデバイスとして機能するためには，コントロール・エンドポイント（EP0）に対して少なくともパソコンから送られてくる種々のコマンドのうちの「標準デバイス・リクエスト」と呼ばれるコマンド群に応答できなければなりません．これらは主としてデバイス固有の情報（機能の概要やエンドポイントの数など）を要求したり，USBツリー内のアドレスを設定したり，動作モードを切り替えたり，といった機能を持っています．これらの処理は**図3**左下のデバイス・リクエスト処理に相当し，usb.devicelayer.c，usb.descriptors.hに記述されています．これらのコマンド群はコントロール・トランスファと呼ばれるプロトコルに従って伝達，実行，ステータス報告が行われます．コントロール・トランスファの処理はusbn960x.cに記述されています．

● 標準USBディスクリプタ

デバイスは，自分の固有情報を標準USBディスクリプタというデータ・テーブル群の形で保持しています．これは標準デバイス・ディスクリプタ，標準コンフィグレーション・ディスクリプタ，標準インターフェース・ディスクリプタ，標準エンドポイント・ディスクリプタと呼ばれる4種類のデータ・テーブルの組み合わせで構成されています．

標準デバイス・ディスクリプタには，デバイス全体の情報，例えばUSBのサポート・バージョンやデバイス機能の概略（Class Codeなど），製造者名，デバイス名などが記載されています．

標準エンドポイント・ディスクリプタには，該当エンドポイントの番号，データ転送方式と方向，最大のパケット・サイズなどが記載されています．USBでは一つのデバイスに複数の独立した機能を実装することができます．その機能ごとにエンドポイントの集合体であるインターフェースが定義されます．

標準インターフェース・ディスクリプタには，該当インターフェースの機能の概略（Class Codeなど）が記載されています．また，USBではどの機能（インターフェース）を有効にするか，動的に変更するための機構が用意されています．

どの機能群を有効にするか，という組み合わせをコンフィグレーションと呼び，標準コンフィグレーション・ディスクリプタにその概略が記載されています．パソコン側はGet Descriptor標準デバイス・リクエストを使用して，用意されているコンフィグレーションのディスクリプタをすべて読み出し，Set Configuration標準デバイス・リクエストで希望のコンフィグレーションを選択することになります．

ちなみに，標準インターフェース・ディスクリプタ，標準コンフィギュレーション・ディスクリプタそのものには，それぞれどのエンドポイントやインターフェースが属するかという情報は記載されていません．Get Descriptor標準デバイス・リクエストでディスクリプタを要求されたときに，所属するエンドポイントおよびインターフェースのディスクリプタをすべていっしょに送ることで，この情報をパソコン側に知らせることになります．ちなみに，同時に有効にならない複数のインターフェースは，同じエンドポイントを所有することができます．そのような場合，インターフェースによって同一エンドポイントのデータ転送方式や方向が異なっていてもかまいません．

パソコンはUSBデバイスの接続を検知すると，バス・リセットをかけ，Set Address標準デバイス・リクエストによりデバイスにUSBアドレスを割り当て，Set Configuration標準デバイス・リクエストによりインターフェースをアクティベートし，デバイスを動作可能状態に持っていきます．

● Mass Storage Classへの対応

デバイスがMass Storage Classとして機能するためには，以下のことが必要です．

1) Mass Storage Class機能専用のインターフェースを用意すること
2) そのインターフェースのディスクリプタにデバイスがサポートするデータ転送方法とコマンド・コード体系を正しく記述しておくこと
3) Mass Storage Class特有のデバイス・リクエストをサポートすること

1) Mass Storage Class機能専用のインターフェース

Mass Storage Classの概要文書[8]では，Mass Storage Class用インターフェースを2種類定義しています．すなわち，Control/Bulk/Interrupt protocol (CBI) とBulk-Only Transportです．

前者は「Full-speedのUSBフロッピー・ディスクのみで用いること，新規設計には用いないことを推奨する」と書かれているので，Bulk-Only Transport[9]を用いることにします．また，コマンド・コード体系[8]については6種類定義されています．そのうちUSBフロッピー・ディスクに適したものは，UFI，SFF-8070iの二つです．

今回はUSB-IFから正式なspecificationを入手できるUFI[10]を選択しました．Bulk-Only Transport用のインターフェースは，Bulk-Inエンドポイント(Bulkトランザクションによって通信するIN方向のエンドポイント)，Bulk-Outエンドポイントがそれぞれ一つずつで構成されます．

2) ディスクリプタ

ディスクリプタについては，本機に搭載する機能はMass Storage Class機能のみなので，インターフェースもこの機能用のもの1種類だけです．したがって，コンフィグレーションも1種類しか必要ありません．

本機に用意するディスクリプタは，標準デバイス・ディスクリプタが一つ，標準コンフィグレーション・ディスクリプタが一つ，標準インターフェース・ディスクリプタが一つ，標準エンドポイント・ディスクリプタが二つ(Bulk-Out，Bulk-In用が一つずつ)，となります．

Mass Storage Classに対応したデバイスであることをパソコンに知らせるためには，標準デバイス・ディスクリプタの5バイト目(bDeviceClass)を00h，標準インターフェース・ディスクリプタの6バイト目(bInterfaceClass)を08hとする必要があります．さらに，Bulk-Only Transportであることは同ディスクリプタの8バイト目(bInterfaceProtocol)を50hとすることで，UFIコマンド体系を採用していることは7バイト目(bInterfaceSubClass)を04hとすることで記述します．

3) Mass Storage Class特有のデバイス・リクエスト

デバイス・リクエストは，Bulk-Only Mass Storage ResetリクエストとGet Max LUNリクエストです[9]．前者はMass Storage Class用インターフェース関連の機能のリセットで，後者はデバイスのLogical Unit Number (LUN) の最大値を応答します．

LUNは一つの物理デバイスに複数の(サブ)デバイスを用意するための機構です(例えば，マルチ光学ドライブ，LUN0はCD-ROMドライブ機能，LUN1はDVD-RAMドライブ機能)．データ用パーティションをたくさん(3個以上)用意したい場合にはLUNを用いる必要がありますが，今回の設計では必要ないため，このコマンドに対して0を返します．

● Mass Storage Classのための処理

Mass Storage Classとしての機能は，**図3**の中ほどの下の部分で処理しています．コードはboms.cbwhandler.cに，UFIの処理はboms.ufiproc.cに記述しています．

Bulk-Only Transport用のインターフェースはBulk-In，Bulk-Outエンドポイントの各一つずつで構成されます[9]．パソコン側はデバイスのBulk-Outエンドポイントに対して(もちろんBulk-Outトランザクションを用いて)，Command Block Wrapper (CBW) と呼ばれるデータ・ブロックを送ります．CBWは，使用するコマンド体系のコマンド(今回はUFIコマンド)のほかに，データ転送量(dCBWDataTransferLength)やその方向(bmCBWFlagsのビット7)などの情報を含んでいます．デバイス側はこのCBW，およびCBWに含まれるUFIコマンドを解釈し，必要に応じてBulk-Out/Inエンドポイント経由でデータの送受信を行い，最後にCommand Status Wrapper (CSW) と呼ばれるデータ・ブロックをBulk-Inエンドポイントからパソコンに送出します．CSWにはコマンドの実行ステータス(bCSWStatus)と処理されなかったデータ量(dCSWResidue)が含まれています．

Bulk-Only Transportでは，かならずこの順にデータがやりとりされる必要があり，いったんパソコン側がCBWを発行すれば，デバイス側がCSWを返すか，パソコン側

がBulk-Only Mass Storage Resetコマンドでリセットをかけるまで，次のCBWが発行されることはありません．もしこの規則に従わないタイミングでCBWが発行されたり，CBWの内容に不備がある場合，デバイスはBulk-Out, Bulk-In両方のエンドポイントをストールさせ，Bulk-Only Mass Storage Resetデバイス・リクエストを待ちます．

後述のようにエンドポイントのストールを検知したパソコン側は，まずClear Feature標準デバイス・リクエストによってエンドポイントをリセットしようとしますが，この場合はこのデバイス・リクエストにはエラーを返します．これにより，パソコン側はCBWのエラーであると認識し，Bulk-Only Mass Storage Resetデバイス・リクエストを発行してくることになります．

CSWに含まれる実行ステータスは，正常終了，失敗，CBW不整合の3種類があります．失敗はUFIコマンドの実行エラーを意味します．CBW不整合はCBW内に記述されたデータ転送量と転送方向がUFIコマンドで指定されたものと矛盾することを意味します．例えば，CBWにはデータ転送がOUT方向と指定されているのに，UFIで必要なデータ転送がIN方向というような場合です．参考文献(9)の「6.7 The Thirteen Cases(pp.18-20)」に詳しく定義されています．この資料中では，host expectations(ホストが期待するデータ転送量と転送方向)がCBWで指定されたデータ転送量と転送方向を意味し，device intent(デバイスの意図するデータ転送量と転送方向)がUFIで指定されたデータ転送量と転送方向を意味します．

CBW不整合の判定にはいくつかの選択肢がありますが，本機ではデータ転送が必要な場合で，

i) 方向が反対の場合
ii) 方向は一致するが，CBWで指定されたデータ量のほうがUFIで指定されたデータ量よりも小さい場合

という基準で判定しています．また，方向が一致してCBWで指定されたデータ量がUFIで指定されたデータ量よりも多い場合，デバイスは(USBでいう)IN方向なら足りない分だけ0を追加し，OUT方向なら余分なデータを読み捨てます．そして，その量をCSWの未処理データ数(dCSW-Residue)にセットします．CBW不整合と判断したデバイスは，CBWで指定されたデータ転送方向がIN方向の場合はBulk-Inエンドポイントを，OUT方向の場合は両方のエンドポイントをストールさせます．パソコン側はエンドポイントがストールしたのを検知すると，Clear Feature標準デバイス・リクエストで該当エンドポイントをリセットし，CSWを読み出します．もちろん，このCSWのbCSW-StatusにはCBW不整合が発生したことが記述されていることになります．

留意すべき点は，CBW不整合によるエラーを報告しない場合はかならずCBWに指定されている量のデータ送受信を行うことです．読み捨てたり0を追加したりした量は，CSWのdCSWResidueにセットします．これはUFIコマンドの実行に失敗した場合にもあてはまります．例えば，データやパラメータを読み出すようなUFIコマンドに失敗し，送るべきデータが用意できなかった場合は，CBWで指定された分だけ0を送ります．

● UFIコマンド群の処理

UFIコマンド群を**表1**[10]に示します．これらの処理はboms.ufiproc.cに記述されています．本機はフロッピー・ディスクをエミュレートしているので，*マークの付いたコマンドは，実際には何もせず正常終了を返すだけです．

UFIコマンドにおいて，すべてのデータはBlock単位でアクセスされます．1 Blockは1セクタに相当し，本機では一般的なフロッピー・ディスクと同じ512バイトとしています．Mode SenseコマンドやRead Format Capacitiesで要求されるドライブやメディア固有の情報は2DD(720K

表1 UFIコマンド群

コマンド名	コード	データ転送方向(パソコン側から見て)	参照[10]
*Format Unit	04h	OUT	p.16
Inquiry	12h	IN	p.19
*Mode Select	55h	OUT	p.21
Mode Sense	5Ah	IN	p.22
Prevent-Allow Medium Removal	1Eh	なし	p.29
Read(10)	28h	IN	p.30
Read(12)	A8h	IN	p.31
Read Capacity	25h	IN	p.32
Read Format Capacities	23h	IN	p.33
Request Sense	03h	IN	p.37
*Rezero	01h	なし	p.39
*Seek(10)	2Bh	なし	p.40
*Send Diagnostic	1Dh	なし	p.41
*Start-Stop Unit	1Bh	なし	p.42
*Test Unit Ready	00h	なし	p.44
*Verify	2Fh	なし	p.45
Write(10)	2Ah	OUT	p.46
Write(12)	AAh	OUT	p.47
Write and Verify	2Eh	OUT	p.48

8

バイト，512バイト/セクタ，9セクタ/トラック，2ヘッド，80トラック）のみサポートしたフロッピー・ディスク・ドライブとして答えています．Read Capacityは，Blockの大きさ（＝セクタ長）と総Block数－1を要求するコマンドですが，これに対しては正確に応答します．つまり，1セクタ512バイトで64Kバイトなので，総Block数－1は127（7Fh）です．

Prevent-Allow Medium Removalコマンドに対しては，メディア・ロック不可能であることを示すために，ロック要求の場合のみエラーを返します．

Request Senseコマンドでは，直前に実行されたコマンドの終了状態を返します．必要となるのは，Sense Key（主エラー・コード）が05hのものだけなので簡単です．

● パーティション・テーブル

パーティション・テーブルは，マスタ・ブート・レコード（MBR記憶領域の最初の1セクタ）の1BEhバイト目から1FFhバイト目までに配置されたテーブルで，パーティション情報を記述します(11), (12)．各16バイトのエントリ四つで構成され，**表2**に示すように各エントリがパーティションの開始点，終了点，容量を保持しています．開始点と終了点はCHS表記になっていますが，最近のOSでは最後の8バイト分，開始点のLBAアドレスとセクタ長のみを参照していると思われます．LBAアドレスは最初のセクタを0番として数えた該当セクタの番号です．また，Partition Typeは00h（何もなし）としてあります．

表2　パーティション・テーブル

オフセット	内　容	大きさ
1BEh	パーティション1のエントリ	16バイト
1CEh	パーティション2のエントリ	16バイト
1DEh	パーティション3のエントリ	16バイト
1EEh	パーティション4のエントリ	16バイト
1FEh	下位から55h，AAh（テーブルが有効であることを示す）	2バイト

（a）全体構造

オフセット	内　容	大きさ
00h	ブート可能なパーティションなら80h，それ以外なら00h	1バイト
01h	パーティション開始点のヘッド番号	1バイト
02h	パーティション開始点のセクタ番号（ビット5～0）とシリンダ番号の上位2ビット（ビット7～6）	1バイト
03h	パーティション開始点のシリンダ番号の下位8ビット	1バイト
04h	パーティション・タイプ．参考文献(13)参照	1バイト
05h	パーティション終了点のヘッド番号	1バイト
06h	パーティション終了点のセクタ番号（ビット5～0）とシリンダ番号の上位2ビット（ビット7～6）	1バイト
07h	パーティション終了点のシリンダ番号の下位8ビット	1バイト
08h	パーティション開始点のLBAアドレス（最初から数えて何番目のセクタか－1）	4バイト
0Ch	パーティションのセクタ・サイズ	4バイト

（b）エントリの構造

4．まとめ

今回は，特別なドライバなしで制御できるUSB機器の実装例として，アナログ・データ・キャプチャを製作しました．POSIX準拠の環境では`open()`，`close()`，`write()`，`read()`を用いてプログラミングできますが，ddコマンドが用意されている環境ならば，これだけでも制御できます．例えば，デバイス・ファイルが/dev/sdaの場合にCH0，CH1の両チャネルでデータをキャプチャし，パソコンに転送するには，

```
# dd bs=1 count=3 of=/dev/sda1
GG*
読み込んだブロック数は3+0
書き込んだブロック数は3+0
#
# dd bs=1 count=2 if=/dev/sda1
##読み込んだブロック数は2+0
書き込んだブロック数は2+0
#
# dd bs=512 count=32 if=/dev/sda2
                          of=ch0data.dat
読み込んだブロック数は32+0
書き込んだブロック数は32+0
#
# dd bs=512 count=32 if=/dev/sda3
                          of=ch1data.dat
読み込んだブロック数は32+0
書き込んだブロック数は32+0
#
#
```

のように操作します．この処理を行うperlスクリプトを用意しておきました（capt.pl）．もちろん，後ろ二つのコマンドの代わりに，

```
# cat /dev/sda2 > ch0data.dat
# cat /dev/sda3 > ch1data.dat
```

とすることもできます．チャネル数を増やしたい場合は拡張パーティションを用いたり，Logical Unitを増設して見かけ上のユニット数を増やすなどの方法もありますが，かならずしも各チャネルに一つずつパーティションを与える必要があるわけではありません(上のようにcatコマンドで読み出すことはできなくなるが…)．あらかじめ決めておいた位置から読み出せばよいだけです．

本機の動作確認はFedora Core 4とVine Linux 3.2の2種類のLinuxで行っています．開発期間の制約もあり，Windows環境における動作チェックは行っていません．標準デバイス・ディスクリプタでいい加減なベンダID，プロダクトID，プロダクト名をかたっているので，おそらくドライバの自動選択機構のあたりでつまづくのではないかと思っています．認識さえされれば，Windows NT系列はPOSIXに準拠しているので，内部デバイス(UNIXのデバイス・ファイルに相当)に同じシステム・コールを用いてアクセスすることができるはずです．また，cygwinなどのUNIX互換環境をインストールすれば，ddコマンドを用いたり，用意したスクリプトでデバイスを制御できると思われます．

個人的な事情で恐縮ですが，幸運にも今回のADuC7026の開発ソフトウェアなど，限られた数のアプリケーション以外ではWindowsを使用しなくてよい環境にいるため，現時点ではWindows環境で動作させるための作業を行う予定はありません．もちろん，多くのご要望をいただいた場合は努力します．また，GPLですので，これに準拠する限りプログラムは自由に改変，再配布していただいてけっこうです．

バグ・フィクスやアップデートなどが出た場合は，筆者のホームページ(http://www.digicat.info/dwm/)にてお知らせいたします．

参考・引用*文献

(1) Analog Devices；ADuC7019/20/21/22/24/25/26/27 Data sheet, Rev.0, Nov. 2005.
(2) National Semiconductor；USBN9603/USBN9604 Univeral Serial Bus Full Speed Node Controller with Enhanced DMA Support, Rev.1.3, http://www.national.com/pf/US/USBN9603.html, Jun. 2003.
(3) Winbond；W24257A Datasheet, Rev.A6, http://www.winbond.com.tw/E-winbondhtm/partner/PDFresult.asp?Pname＝715, Jan. 1998.
(4) USB通信プログラミングテクニック, http://www.picfun.com/usbframe.html.
(5) 桑野雅彦；USBコントローラUSBN9602を使ったUSBターゲット機器の開発，USBハード&ソフト開発のすべて，CQ出版社，TECHI Vol. 8, pp.43-54.
(6) 桑野雅彦；USBコントローラUSBN9602/9603/9604の使いかた，CQ出版社，TECHI Vol. 27, pp.60-69.
(7) Compaq, Intel, Microsoft and NEC；Universal Serial Bus Specification, Rev. 1.1, http://www.usb.org/developers/docs/usbspec.zip, Sep. 1998.
(8) USB-IF；Universal Serial Bus Mass Storage Class Specification Overview, Rev.1.2, http://www.usb.org/developers/devclass_docs/usb_msc_overview_1.2.pdf, Jun. 2003.
(9) USB-IF；Universal Serial Bus Mass Storage Class Bulk-Only Transport, Rev. 1.0, http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf, Sep. 1999.
(10) USB-IF；Universal Serial Bus Mass Storage Class UFI Command Specification, Rev.1.0, http://www.usb.org/developers/devclass_docs/usbmass-ufi10.pdf, Dec. 1998.
(11) Pierre's field guide to partiion table recovery, http://www.datarescue.com/laboratory/partition.htm
(12) Minimal partition table specification, 2.2 Partition descriptor, http://www.win.tue.nl/~aeb/partitions/partition_tables-2.html
(13) Minimal partition table specification, 1. List of partition identifiers for PCs, http://www.win.tue.nl/~aeb/partitions/partition_types-1.html

8

こうの・たかし
科学技術振興機構ERATO合原プロジェクト

<筆者プロフィール>
1996年 東京大学医学部医学科卒．同年 東京大学工学系研究科計数工学専攻博士課程入学．2002年 工学博士号取得．同年 浜松医大医療情報部医員．2004年 現職へ．M.D.Ph.D.学生時代よりFPGAを用いた回路設計に携わる．現在の研究テーマはNeuromorphic Hardware．